Large-Scale Gaussian Process Regression via Doubly Stochastic Gradient Descent

Xinyan Yan, Bo Xie, Le Song, Byron Boots {XINYAN.YAN, BXIE33, LSONG, BBOOTS}@CC.GATECH.EDU
College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332

Abstract

Gaussian process regression (GPR) is a popular tool for nonlinear function approximation. Unfortunately, GPR can be difficult to use in practice due to the $O(n^2)$ memory and $O(n^3)$ processing requirements for n training data points. We propose a novel approach to scaling up GPR to handle large datasets using the recent concept of doubly stochastic functional gradients. Our approach relies on the fact that GPR can be expressed as a convex optimization problem that can be solved by making two unbiased stochastic approximations to the functional gradient, one using random training points and another using random features, and then descending using this noisy functional gradient. The effectiveness of the resulting algorithm is evaluated on the well-known problem of learning the inverse dynamics of a robot manipulator.

1. Introduction & Related Work

Gaussian processes (GPs) (Rasmussen & Williams, 2006) have been successfully applied to a range of learning problems in a wide variety of disciplines, including object categorization in computer vision (Kapoor et al., 2010), natural language processing in computational linguistics (Cohn et al., 2014), and robotics (Williams & Rasmussen, 1996). However, exact inference in Gaussian process requires $O(n^3)$ computation time and $O(n^2)$ storage for a dataset with n samples. The steep computational requirements for GPs and other kernel-based methods have inspired numerous recent attempts to scale up kernel-based machine learning from both approximation and optimization perspectives.

Approximation methods include rank-r ($r \leq n$) approximation of the kernel matrix (e.g., (Williams & Seeger, 2000; Smola & Schölkopf, 2000), and finite approximation of the kernel function space by using r random features (e.g., (Rahimi & Recht, 2008)). These approaches reduce computation time and memory demand to $O(nr^2)$ and $O(nr)$ respectively. However, without further assumptions on the regularity of the kernel matrix, the generalization ability after approximation is typically of the order $O(1/\sqrt{r} + 1/\sqrt{n})$ (Drineas & Mahoney, 2005; Lopez-Paz et al., 2014), which implies that r needs to be $O(n)$ to retain similar performance.

In contrast to approximation approaches, optimization schemes often attempt to address scalability issues through *iterative* methods. For example, (block) coordinate descent with parameter block size r (e.g., (Shalev-Shwartz & Zhang, 2013)), or functional gradient descent with batch size r (e.g., (Kivinen et al., 2004)) both require $O(nr^2)$ time and $O(nr)$ space at each iteration. A serious drawback of these approaches is that although *learning* is fast, many data points are retained for prediction.

Several approaches closely related to GPs have also been developed in the robotics and AI communities, where work has focused on learning robot kinematics or dynamics, or representing value functions (D'Souza et al., 2001; Nguyen-Tuong et al., 2008; Konidaris et al., 2011; Buck et al., 2002). An interesting application that has inspired many of these algorithms is the problem of learning inverse dynamics from large datasets. Examples include Locally Weighted Regression (LWR) (Atkeson et al., 1997) and Locally Weighted Projection Regression (LWPR) (Vijayakumar & Schaal, 2000) However, LWPR has a large number parameters that are notoriously difficult to tune and have significant impact on both accuracy and computation time. Incremental Local Gaussian Regression (I-LGR) (Meier et al., 2014) attempts to improve the accuracy of LWPR by using GPs as local models. Despite their computational efficiency, the accuracy of these methods is often inferior to Gaussian process regression and approximation methods such as Sparse Spectrum Gaussian Process Regression (SSGPR) (Gijsberts & Metta, 2012).

In this paper, we develop a new method for scaling Gaussian process regression (GPR) (Williams & Rasmussen, 1996) up to large datasets (millions of training data samples) using the recent concept of *Doubly Stochastic Gradient Descent* (DSGD) proposed by Dai et.al. (Dai et al.,

2014). Our approach relies on the fact that GPR can be expressed as a convex optimization problem that can be solved by making two unbiased stochastic approximations to the functional gradient, one using random training points and another using random features, and then descending using this noisy functional gradient. Gaussian process regression via doubly stochastic gradient descent (GPR-DSGD) aims to achieve a better balance of computation, memory, and accuracy than previous approximation methods by leveraging enormous sets of random features while simultaneously scaling to massive datasets. Unlike the nonlinear function approximation methods such as LWPR, LGR, and LGP, the resulting algorithm closely approximates the global Gaussian process, achieving higher prediction accuracy while requiring fewer manually tuned parameters. The effectiveness of the resulting algorithm is evaluated on the problem of learning the inverse dynamics of a robot manipulator. The results show that by using huge features sets (up to $\sim 65,000$ random features) GPR-DSGD successfully scales GPR to handle millions of data points with little degradation in accuracy.

2. Gaussian Process Regression

Probabilistic regression estimates the distribution of function values $\boldsymbol{f}_*$ at test points $\boldsymbol{x}_*$, given a training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | \boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}, i = 1, ..., n\}$ of n pairs of input $\boldsymbol{x}_i$ and noisy target y_i. Usually, the noisy target y_i is assumed to be the unobserved function value at $\boldsymbol{x}_i$ plus some additive, independent Gaussian noise ε (Eq. 1). Gaussian process regression is a Bayesian approach that performs inference in the function space with respect to a prior distribution of functions described in Eq. 2.

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma_n^2) \tag{1}$$

$$f(\boldsymbol{x}) \sim \mathcal{GP}(0, k(\boldsymbol{x}, \boldsymbol{x}')), \tag{2}$$

where the covariance function $k : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is a symmetric positive definite (PD) kernel function. All vectors are column vectors and matrices and vectors are bold.

A Gaussian process prior distribution on $f(\boldsymbol{x})$ allows us to encode assumptions on the smoothness of the latent function. The prior joint distribution of the noisy targets in the training dataset and the predictive function value at one test point is:

$$\begin{bmatrix} \boldsymbol{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{G} & \boldsymbol{k_*} \\ \boldsymbol{k_*^\intercal} & k_* \end{bmatrix}\right) \tag{3}$$

with

$$\boldsymbol{y} = [\ y_1 \quad ... \quad y_n\ ]^\intercal, \quad f_* = f(\boldsymbol{x}_*), \quad \boldsymbol{G} = \boldsymbol{K} + \sigma_n^2 \boldsymbol{I}$$
$$\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j), \quad \boldsymbol{k}_* = [k(\boldsymbol{x}_*, \boldsymbol{x}_i)]_{i=1}^n, \quad k_* = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$$

Through conditioning, the posterior distribution of the function value at the test point can be derived as:

$$f_* | \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_* \sim \mathcal{N}\left(\boldsymbol{k}_*^\intercal \boldsymbol{G}^{-1} \boldsymbol{y},\ k_* - \boldsymbol{k}_*^\intercal \boldsymbol{G}^{-1} \boldsymbol{k}_*\right) \tag{4}$$

3. GPR via Functional Gradient Descent

Exact inference in Gaussian processes (Eq. 4) has significant practical limitations in learning and inference on large datasets, due to $O(n^3)$ computation time and $O(n^2)$ storage requirements of the kernel machinery. However, by viewing Gaussian process regression as ridge regression in a reproducing kernel Hilbert space (RKHS), we can use functional gradient descent to iteratively find the solution.

Ridge regression in the RKHS is achieved by searching for a function in the RKHS that minimizes an objective formulated as a functional $R : f \mapsto \mathbb{R}$:

$$R(f) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f(\boldsymbol{x_i})\right)^2 + \frac{\nu}{2} \|f\|_{\mathcal{H}}^2 \tag{5}$$

The first term in the objective measures how well the function fits the training set, while the second term is a regularizer that controls the complexity of the function.

We can solve the problem in Eq. 5 by functional gradient descent. The functional gradient $\nabla R(f)$ is defined as the linear term in the approximation to a perturbed functional:

$$R(f + \epsilon g) = R(f) + \epsilon \langle \nabla R(f), g \rangle_{\mathcal{H}} + O(\epsilon^2). \tag{6}$$

Using the chain rule for functional gradients, and the fact that the gradient of an evaluation functional that evaluates f at a specific point $\boldsymbol{x}$ is $k(\boldsymbol{x}, \cdot)$, the gradient of R is:

$$\nabla R(f) = \sum_{i=1}^{n} \left(f(\boldsymbol{x}_i) - y_i\right) k(\boldsymbol{x}_i, \cdot) + \nu f \tag{7}$$

Therefore, functional gradient descent produces a sequence of intermediate solutions f_t with the update equation:

$$f_{t+1} = f_t - \eta_t \nabla R(f_t), \tag{8}$$

where η_t is the step size. By writing Eq. 8 as update on the coefficients of kernel functions, we can compute the updated $\alpha_{i,t+1}$ by:

$$\alpha_{i,t+1} = (1 - \eta_t \nu)\, \alpha_{i,t} - \eta_t \left(f(\boldsymbol{x}_i) - y_i\right),\ i = 1, ..., n \tag{9}$$

where

$$f_t(\cdot) = \sum_{i=1}^{n} \alpha_{i,t} k(\boldsymbol{x_i}, \cdot). \tag{10}$$

From the posterior distribution of the function value at a test point $\boldsymbol{x}_*$ (Eq. 4), we can see that computing the predictive variance of Gaussian processes requires computing the quantity: $\boldsymbol{k}_*^\intercal \left(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}_*$, which can be evaluated:

$$\begin{aligned} \boldsymbol{k}_*^\intercal \left(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}_* &= k(\boldsymbol{x}_*, \cdot)^\intercal \boldsymbol{\phi} \left(\boldsymbol{\phi}^\intercal \boldsymbol{\phi} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{\phi}^\intercal k(\boldsymbol{x}_*, \cdot) \\ &= k(\boldsymbol{x}_*, \cdot)^\intercal \boldsymbol{\phi} \boldsymbol{\phi}^\intercal \left(\boldsymbol{\phi} \boldsymbol{\phi}^\intercal + \sigma_n^2 \boldsymbol{I}\right)^{-1} k(\boldsymbol{x}_*, \cdot) \end{aligned}$$

where $\boldsymbol{\phi} = [k(\boldsymbol{x}_1, \cdot), \ldots, k(\boldsymbol{x}_n, \cdot)]$, and the second equality relies on the *matrix inversion lemma* (Boyd & Vandenberghe, 2004). Therefore, we just need to estimate the operator:

$$\mathcal{A} = \widehat{\mathcal{C}}\left(\widehat{\mathcal{C}} + \frac{\sigma_n^2}{n} I\right)^{-1} \tag{11}$$

where $\widehat{\mathcal{C}} = \frac{1}{n}\sum_{i=1}^{n} k(\boldsymbol{x}_i, \cdot) \otimes k(\boldsymbol{x}_i, \cdot) = \frac{1}{n}\boldsymbol{\phi}\boldsymbol{\phi}^\intercal$ is the empirical covariance operator. Estimating $\mathcal{A}$ can be accomplished by solving the following convex optimization problem:

$$\min_{\mathcal{A}} \; R(\mathcal{A}) = \frac{1}{2n}\sum_{i=1}^{n} \|k(\boldsymbol{x}_i, \cdot) - \mathcal{A}\, k(\boldsymbol{x}_i, \cdot)\|_{\mathcal{H}}^2 + \frac{\sigma_n^2}{2n}\|\mathcal{A}\|_{HS}^2$$

where $\|\cdot\|_{HS}$ is the Hilbert-Schmidt norm (or generalized Frobenius norm) of the operator. The gradient of $R(\mathcal{A})$ with respect to the operator $\mathcal{A}$ is:

$$\begin{aligned}\nabla R(\mathcal{A}) &= \frac{1}{n}\sum_{i=1}^{n}\Big((\mathcal{A}\, k(\boldsymbol{x}_i, \cdot) - k(\boldsymbol{x}_i, \cdot)) \otimes k(\boldsymbol{x}_i, \cdot)\Big) + \frac{\sigma_n^2}{n}\mathcal{A} \\ &= \mathcal{A}(\widehat{\mathcal{C}} + \frac{\sigma_n^2}{n} I) - \widehat{\mathcal{C}}\end{aligned}$$

The optimal solution is given by the condition $\nabla R(\mathcal{A}) = 0$, thus one can obtain the desired operator via functional gradient descent.

$$\mathcal{A}_{t+1} = \mathcal{A}_t - \xi_t \nabla R(\mathcal{A}_t) \tag{12}$$

where ξ_t is the step size.

4. GPR via Approximate Functional Gradient Descent with Doubly Stochastic Gradients

There are two drawbacks with the functional gradient descent approach: 1) it does not scale to large datasets since each iteration requires one pass through the dataset, and 2) the entire training dataset must be retained in order to evaluate function values on new test data. To overcome these problems, we leverage *doubly stochastic functional gradients* to approximate the functional gradient with two unbiased stochastic approximations (Dai et al., 2014): stochastic gradient descent and random features. Specifically, the approximation at iteration t is

$$f_t = \sum_{i=1}^{t} \alpha_{i,t}\phi_{\boldsymbol{\omega}_i}(\boldsymbol{x}_i)\phi_{\boldsymbol{\omega}_i}(\cdot) = \sum_{i=1}^{t}\beta_{i,t}\phi_{\boldsymbol{\omega}_i}(\cdot), \tag{13}$$

where $\beta_{i,t} = \alpha_{i,t}\phi_{\boldsymbol{\omega}_i}(\boldsymbol{x}_i)$, and $\phi_{\boldsymbol{\omega}_i}(\cdot)$ is a function parameterized by $\boldsymbol{\omega}_i$: $\phi_{\boldsymbol{\omega}_i}(\boldsymbol{x}) = \sqrt{2}\cos(\boldsymbol{\omega}_i^\intercal \boldsymbol{x} + b)$.

Thus the gradient descent update equation for the coefficients β_i's at iteration t becomes:

$$\beta_{t+1,t+1} = -\eta_t(f(\boldsymbol{x}_t) - y_t)\phi_{\boldsymbol{\omega}_t}(\boldsymbol{x}_t) \tag{14}$$

$$\beta_{i,t+1} = (1 - \eta_t\nu)\beta_{i,t}, \quad \text{for } i = 1, ..., t \tag{15}$$

Note that there's still the need to store all the $\boldsymbol{\omega}_i$'s, which take the same amount of memory as $\boldsymbol{x}_i$'s. However, since $\boldsymbol{\omega}_i$ are *pseudo-random* numbers, we can just save the random seeds and regenerate them on-the-fly.

Algorithm 1 $\{\beta_i\}_{i=1}^t$ = Train($\mathbb{P}(\boldsymbol{x}, y)$)

Require: $\mathbb{P}(\boldsymbol{\omega})$, $\boldsymbol{s}$, $\boldsymbol{\eta}$, σ_f, ν, z, r
for $i = 1$ **to** t **do**
  $\boldsymbol{X} = [\boldsymbol{x}_1 ... \boldsymbol{x}_z]^\intercal$, $\boldsymbol{y} = [y_1 ... y_z]^\intercal$, $(\boldsymbol{x}_k, y_k) \sim \mathbb{P}(\boldsymbol{x}, y)$
  // Sample new features
  $\boldsymbol{\phi}$ = GetFeatures($\boldsymbol{X}$, s_i)
  $\hat{\boldsymbol{y}} = \boldsymbol{0}_{b\times 1}$
  // Predict using previous features
  for $j = 1$ **to** $i - 1$ **do**
    $\boldsymbol{\phi}_t$ = GetFeatures($\boldsymbol{X}$, s_j)
    $\hat{\boldsymbol{y}} = \hat{\boldsymbol{y}} + \boldsymbol{\phi}_t\boldsymbol{\beta}_j^\intercal$
  end for
  // Update coefficients
  $\boldsymbol{\Sigma} = \boldsymbol{\phi}^\intercal\boldsymbol{\phi}$
  $\boldsymbol{\beta}_i = -\eta_i\boldsymbol{\Sigma}^{-1}\boldsymbol{\phi}^\intercal(\hat{\boldsymbol{y}} - \boldsymbol{y})$
  $\boldsymbol{\beta}_j = (1 - \eta_i\nu)\boldsymbol{\beta}_j$, for $j = 1, ..., i - 1$
end for

Algorithm 2 $\boldsymbol{\phi}$ = GetFeatures($\boldsymbol{X}$, s)

Require: $\mathbb{P}(\boldsymbol{\omega})$, σ_f, z
$\boldsymbol{\Omega} = [\ \boldsymbol{\omega}_1 \quad ... \quad \boldsymbol{\omega}_r\ ]^\intercal, \boldsymbol{\omega}_i \sim \mathbb{P}(\boldsymbol{\omega})$, with seed s
$\boldsymbol{b} = [\ b_1 \quad ... \quad b_r\ ]^\intercal, b_i \sim \mathcal{U}(0, 2\pi)$, with seed s
$\boldsymbol{\phi} = \frac{\sqrt{2}\sigma_f}{\sqrt{r}}\cos(\boldsymbol{X}\boldsymbol{\Omega}^\intercal + \boldsymbol{B}^\intercal), \boldsymbol{B} = [\ \boldsymbol{b} \quad ... \quad \boldsymbol{b}\ ]_{r\times z}$

The double stochastic approximation of the functional gradient by random features and stochastic gradient descent, or *doubly stochastic gradient descent*, is a very efficient algorithm for approximating regression in large scale kernel machines. As proved in (Dai et al., 2014), it estimates the optimal function in the RKHS in the rate of $O(1/t)$ both in expectation and with high probability, and achieves a generalization bound of $O(1/\sqrt{t})$. The detailed algorithm applied in practice is presented in Alg. 1 and Alg. 2. To converge more quickly, we take advantage of moderate batch size and pre-conditioning as suggested by Dai et.al. (Dai et al., 2014).

We apply similar ideas to the predictive variance estimation. In each iteration t, we approximate the empirical covariance operator with a single data point $\boldsymbol{x}_t$, which leads to the following update

$$\begin{aligned}\mathcal{A}_t &= \mathcal{A}_{t-1} - \xi_t\left(\mathcal{A}_{t-1}\left(\widehat{\mathcal{C}}_t + \frac{\sigma_n^2}{n} I\right) - \widehat{\mathcal{C}}_t\right) \\ &= \left(1 - \frac{\sigma_n^2}{n}\xi_t\right)\mathcal{A}_{t-1} - \xi_t\left(\mathcal{A}_{t-1}\widehat{\mathcal{C}}_t - \widehat{\mathcal{C}}_t\right)\end{aligned} \tag{16}$$

where $\widehat{\mathcal{C}}_t = k(\boldsymbol{x}_t, \cdot) \otimes k(\boldsymbol{x}_t, \cdot)$. Under this approximation, suppose $\mathcal{A}_0 = 0$, then

$$\mathcal{A}_t = \sum_{i\le j}^{t}\gamma_{ij}k(\boldsymbol{x}_i, \cdot) \otimes k(\boldsymbol{x}_j, \cdot). \tag{17}$$

which can be shown by induction. The base case is $\mathcal{A}_1 = k(\boldsymbol{x}_1, \cdot) \otimes k(\boldsymbol{x}_1, \cdot)$. Assume $\mathcal{A}_{t-1}$ has the bases, then the first term in the update rule keeps all the bases in the $\mathcal{A}_{t-1}$, the term $\mathcal{A}_{t-1}\widehat{\mathcal{C}}_t$ adds additional $\sum_{i=1}^{t-1} k(\boldsymbol{x}_i, \cdot) \otimes k(\boldsymbol{x}_t, \cdot)$ basis, and then $\widehat{\mathcal{C}}_t$ adds $k(\boldsymbol{x}_t, \cdot) \otimes k(\boldsymbol{x}_t, \cdot)$.

To use the doubly stochastic gradient, we approximate $\widehat{\mathcal{C}}_t$ using random features utilizing the following relationship

$$\begin{aligned}\widehat{\mathcal{C}}_t &= \mathbb{E}\left[\boldsymbol{\phi}_{\boldsymbol{\omega}}(\boldsymbol{x}_t)\boldsymbol{\phi}_{\boldsymbol{\omega}}(\cdot)\right] \otimes \mathbb{E}\left[\boldsymbol{\phi}_{\boldsymbol{\omega}}(\boldsymbol{x}_t)\boldsymbol{\phi}_{\boldsymbol{\omega}}(\cdot)\right] \\ &= \mathbb{E}\left[\boldsymbol{\phi}_{\boldsymbol{\omega}}(\boldsymbol{x}_t)\boldsymbol{\phi}_{\boldsymbol{\omega}}(\cdot) \otimes \boldsymbol{\phi}_{\boldsymbol{\omega}'}(\boldsymbol{x}_t)\boldsymbol{\phi}_{\boldsymbol{\omega}'}(\cdot)\right],\end{aligned} \tag{18}$$

where $\boldsymbol{\omega}$ and $\boldsymbol{\omega}'$ are two independent random frequencies. Now $\mathcal{A}_t$ is spanned by the outer products of the cosine functions $\boldsymbol{\phi}_{\boldsymbol{\omega}_i} \otimes \boldsymbol{\phi}_{\boldsymbol{\omega}'_j}$, i.e.,

$$\mathcal{A}_t = \sum_{i \le j}^{t} \theta_{ij} \boldsymbol{\phi}_{\boldsymbol{\omega}_i}(\cdot) \otimes \boldsymbol{\phi}_{\boldsymbol{\omega}'_j}(\cdot). \tag{19}$$

In each iteration t, we draw a pair of independent random features $\boldsymbol{\omega}_t$ and $\boldsymbol{\omega}'_t$ and update $\mathcal{A}_{t-1}$ by scaling its coefficients by $\left(1 - \frac{\sigma_n^2}{n}\xi_t\right)$, adding the new basis $\boldsymbol{\phi}_{\boldsymbol{\omega}_t} \otimes \boldsymbol{\phi}_{\boldsymbol{\omega}'_t}$ (due to $\widehat{\mathcal{C}}_t$) and also bases $\boldsymbol{\phi}_{\boldsymbol{\omega}_i} \otimes \boldsymbol{\phi}_{\boldsymbol{\omega}'_t}$ (due to $\mathcal{A}_{t-1}\widehat{\mathcal{C}}_t$).

The coefficients for the bases due to $\mathcal{A}_{t-1}\widehat{\mathcal{C}}_t$ requires evaluating the current data point $\boldsymbol{x}_t$ at all the previous cosine function. That is,

$$\mathcal{A}_{t-1}\widehat{\mathcal{C}}_t = \sum_{i \le j}^{t-1} \theta_{ij} \boldsymbol{\phi}_{\boldsymbol{\omega}'_j}(\boldsymbol{x}_t)\boldsymbol{\phi}_{\boldsymbol{\omega}'_t}(\boldsymbol{x}_t)\boldsymbol{\phi}_{\boldsymbol{\omega}_i}(\cdot) \otimes \boldsymbol{\phi}_{\boldsymbol{\omega}'_t}(\cdot). \tag{20}$$

The updates for the corresponding coefficients are

$$\begin{aligned}\theta_{ij} &= \left(1 - \frac{\sigma_n^2}{n}\xi_t\right)\theta_{ij},\ i \le j < t \\ \theta_{it} &= -\xi_t \sum_{j \ge i}^{t-1} \theta_{ij} \boldsymbol{\phi}_{\boldsymbol{\omega}'_j}(\boldsymbol{x}_t)\boldsymbol{\phi}_{\boldsymbol{\omega}'_t}(\boldsymbol{x}_t),\ i < t \\ \theta_{tt} &= \xi_t \boldsymbol{\phi}_{\boldsymbol{\omega}_t}(\boldsymbol{x}_t)\boldsymbol{\phi}_{\boldsymbol{\omega}'_t}(\boldsymbol{x}_t)\end{aligned} \tag{21}$$

Note that, compared with the predictive mean, the predictive variance operator requires $O(t^2)$ coefficients, which is computationally more expensive.

After we obtain the estimated operator $\mathcal{A}$, we can calculate the predictive variance by evaluating $\mathcal{A}$ from both sides at a new data point $\boldsymbol{x}$. That is, we can view the operator as a function that takes two arguments $\mathcal{A}(\cdot, \cdot)$, and we can evaluate the function value by putting in the two arguments as $\boldsymbol{x}$ according to Eqn. 19. Thus, the predictive variance can be calculated

$$k(\boldsymbol{x}, \boldsymbol{x}) - \mathcal{A}(\boldsymbol{x}, \boldsymbol{x}). \tag{22}$$

5. Experiments: Learning Inverse Dynamics

Inverse dynamics models are necessary for model-based control tasks (Schaal & Atkeson, 2010), but can be difficult to specify analytically due to nonlinearities induced by friction, inertia, contact, backlash, hydraulics, cable stretch, flexibility, and complex actuator dynamics. Therefore, the problem of modeling inverse dynamics is often cast as a learning problem with the goal of finding a function $\boldsymbol{\tau} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$, which maps joint positions $\boldsymbol{q}$ (rad), velocities $\dot{\boldsymbol{q}}$ (rad/s), and accelerations $\ddot{\boldsymbol{q}}$ (rad/s^2) to torques $\boldsymbol{\tau}$ (Nm).

Joint	I-SSGPR$_{512}$	I-SSGPR$_{4,096}$	LWPR	DSGD
J1	0.019	**0.012**	0.039	0.013
J2	0.015	**0.006**	0.039	0.009
J3	0.010	**0.004**	0.033	0.007
J4	0.003	**0.002**	0.023	0.004
J5	0.021	**0.010**	0.061	0.018
J6	0.023	**0.010**	0.071	0.015
J7	0.007	**0.004**	0.026	0.007
Ave.	0.014	**0.007**	0.038	0.009

Table 1. Sarcos dataset: (One-sweep) Normalized mean squared error (nMSE) of online training data. Number of random features for DSGD: 16,384.

Joint	I-SSGPR$_{512}$	I-SSGPR$_{4,096}$	LWPR	DSGD
J1	0.330	**0.007**	0.041	0.013
J2	0.251	**0.005**	0.032	**0.005**
J3	0.436	0.013	0.060	**0.008**
J4	0.344	0.007	0.044	**0.004**
J5	0.377	0.012	0.044	**0.006**
J6	0.430	0.014	0.060	**0.005**
J7	0.385	0.008	0.041	**0.003**
Ave.	0.365	0.009	0.041	**0.006**

Table 2. KUKA$_1$ dataset: (One-sweep) Normalized mean squared error (nMSE) of online training data. Number of random features for DSGD: 65,536.

We evaluate Gaussian process regression via doubly stochastic gradient descent (GPR-DSGD) by learning a model of inverse dynamics for torque control on a 7-DOF anthropomorphic SARCOS arm (Hollerbach & Jacobsen, 1996) and a 7-DOF KUKA lightweight arm (Bischoff et al., 2010). We compare against two state-of-the-art approaches including incremental sparse spectrum GPR (I-SSGPR) (Gijsberts & Metta, 2012) and locally weighted projection regression (LWPR) (Vijayakumar & Schaal, 2000). For all models, we evaluate the accuracy of the *predictive mean* obtained from our method (GPR-DSGD) under two experiment settings: 1) the *One-sweep* setting, where the training dataset is passed through once (simulating online learning), and 2) the *Multiple-pass* setting, where the training set is traversed several times until the algorithm converges.

A small offline training set is utilized to initialize the parameters. The parameters of I-SSGPR and GPR-DSGD are initialized by maximizing the log-likelihood. For all runs GPR-DSGD batchsize was set to 8,192 (the number of data

Joint	I-SSGPR$_{512}$	I-SSGPR$_{4,096}$	LWPR	DSGD
J1	0.00432	**0.00011**	0.00946	0.00015
J2	0.00566	**0.00088**	0.02713	0.00125
J3	0.00421	0.00007	0.00797	**0.00006**
J4	0.00469	**0.00006**	0.00487	0.00040
J5	0.01898	0.00046	0.01560	**0.00013**
J6	0.03160	0.00059	0.00279	**0.00016**
J7	0.01118	0.00041	0.00115	**0.00014**
Ave.	0.01152	0.00037	0.00762	**0.00032**

Table 3. KUKA$_{\text{sim}}$ dataset: (One-sweep) Normalized mean squared error (nMSE) of online training data. Number of random features for DSGD: 65,536.

points considered in each iteration) and blocksize was set to 4,096 (the number of feature coefficients updated in each iteration). The inverse dynamics of each joint is viewed as independent and learned via separate regressions.

5.1. Single-Sweep Setting

We first evaluated GPR-DSGD in a setting where data is encountered sequentially. We considered three datasets that vary in size, platform, type of motion, and how well the offline training set represents the data encountered during online learning (Meier et al., 2014).

Sarcos: We first evaluate GPR-DSGD on the Sarcos benchmark dataset (Rasmussen & Williams, 2006). All algorithm parameters are initialized on a batch of 4,449 data points. The algorithms then sweep through an additional 44,484 datapoints, updating the models given the new data. I-SSGPR is trained twice, with 512 and 4,096 random features. We report the normalized mean squared error (nMSE) on the online training data and found that GPR-DSDG and I-SSGPR outperform LWPR (Table 1). Despite leveraging a much larger number of random features, GPR-DSDG does not outperform I-SSGPR on the Sarcos benchmark.

KUKA$_1$: The KUKA dataset consists of rhythmic motions of a KUKA arm at various speeds (Meier et al., 2014). The dataset has 17,560 offline training data points consisting of partial coverage of the total range of available speeds in the online data. The online portion consists of 180,360 data points including the full range of speeds. Despite the size of the dataset, GPR-DSDG outperforms the competing approaches by leveraging a *much* larger set of random features (Table 2).

KUKA$_{\text{sim}}$: KUKA$_{\text{sim}}$ is a large dataset of 2 million data points generated by the SL simulator for evaluating inverse dynamics (Meier et al., 2014). In order to match prior work (Meier et al., 2014), we randomly drew 1% of the data points to form a heldout test set. Due to the large number of training data points, all algorithms converge during the single sweep. Again, I-SSGPR and GPR-DSDG outperform LWPR (Table 3).

Joint	I-SSGPR$_{512}$	I-SSGPR$_{4,096}$	LWPR	DSGD
J1	0.020	0.014	0.035	**0.011**
J2	0.016	0.008	0.031	**0.007**
J3	0.011	**0.005**	0.027	**0.005**
J4	0.004	**0.002**	0.016	**0.002**
J5	0.021	**0.012**	0.058	0.013
J6	0.025	0.012	0.068	**0.011**
J7	0.007	**0.004**	0.022	**0.004**
Ave.	0.015	**0.008**	0.037	**0.008**

Table 4. SARCOS dataset: (Multi-pass) Normalized mean squared error (nMSE) on heldout data. Number of random features for DSGD: 16,384. 60 iterations, $\sim$ 12 passes.

Joint	I-SSGPR$_{512}$	I-SSGPR$_{4,096}$	LWPR	DSGD
J1	0.332	0.009	0.040	**0.004**
J2	0.258	0.006	0.020	**0.002**
J3	0.446	0.014	0.025	**0.002**
J4	0.356	0.008	0.017	**0.001**
J5	0.386	0.014	0.013	**0.002**
J6	0.443	0.016	0.016	**0.001**
J7	0.397	0.009	0.013	**0.001**
Ave.	0.374	0.011	0.020	**0.002**

Table 5. KUKA$_1$ dataset: (Multi-pass) Normalized mean squared error (nMSE) on heldout data. Number of random features for DSGD: 65,536. 150 iterations, $\sim$ 8 passes.

5.2. Multi-pass Setting

To further evaluate the accuracy of GPR-DSDG, LWPR, and I-SSGPR after convergence, we learned inverse dynamics in the batch setting. Unlike the single-sweep setting where the test set for Sarcos and KUKA$_1$ are the same as the online training set, 20% of the data is held out, and multiple passes through the training data are carried out until convergence. The parameters used are the same as in the single-sweep setting. The results presented in Table 4 – 5 show that GPR-DSDG consistently outperforms previous approaches by leveraging larger sets of features.

6. Conclusion

We present GPR-DSGD, a novel approach to scaling Gaussian process regression up to large datasets using the recent concept of doubly stochastic functional gradients. Our algorithm relies on the fact that GPR can be expressed as a convex optimization problem that can be solved by making two unbiased stochastic approximations to the functional gradient. This allows us to use large sets of random features and training data. We compare to state-of-the-art nonparametric regression techniques for large datasets and show that our approach scales to millions of samples while maintaining prediction accuracy superior to that of previous methods.

References

Atkeson, C.G., Moore, A.W., and S.Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(73), 1997.

Bischoff, Rainer, Kurth, Johannes, Schreiber, Guenter, Koeppe, Ralf, Albu-Schaeffer, Alin, Beyer, Alexander, Eiberger, Oliver, Haddadin, Sami, Stemmer, Andreas, Grunwald, Gerhard, and Hirzinger, Gerhard. The kuka-dlr lightweight robot arm - a new reference platform for robotics research and manufacturing. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pp. 1–8, June 2010.

Boyd, Stephen and Vandenberghe, Lieven. *Convex Optimization*. Cambridge University Press, Cambridge, England, 2004.

Buck, S., Beetz, M., and Schmitt, T. Approximating the value function for continuous space reinforcement learning in robot control. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pp. 1062–1067 vol.1, 2002. doi: 10.1109/IRDS.2002.1041532.

Cohn, Trevor, Preotiuc-Pietro, Daniel, and Lawrence, Neil. Gaussian processes for natural language processin. *ACL 2014*, 2014.

Dai, B., Xie, B., He, N., Liang, Y., Raj, A., Balcan, M., and Song, L. Scalable kernel methods via doubly stochastic gradients. In *Neural Information Processing Systems*, 2014.

Drineas, P. and Mahoney, M. On the nystr om method for approximating a gram matrix for improved kernel-based learning. *JMLR*, 6:2153–2175, 2005.

D'Souza, Aaron, Vijayakumar, Sethu, and Schaal, Stefan. Learning inverse kinematics. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pp. 298–303. IEEE, 2001.

Gijsberts, Arjan and Metta, Giorgio. Real-time model learning using incremental sparse spectrum gaussian process regression. *Neural Networks*, August 2012.

Hollerbach, John M. and Jacobsen, Stephen C. Anthropomorphic robots and human interactions. In *WASEDA UNIVERSITY*, pp. 83–91, 1996.

Kapoor, Ashish, Grauman, Kristen, Urtasun, Raquel, and Darrell, Trevor. Gaussian processes for object categorization. *International Journal of Computer Vision*, 88(2):169–188, 2010. ISSN 0920-5691. doi: 10.1007/s11263-009-0268-3. URL http://dx.doi.org/10.1007/s11263-009-0268-3.

Kivinen, J., Smola, A. J., and Williamson, R. C. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8), Aug 2004.

Konidaris, G.D., Osentoski, S., and Thomas, P.S. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pp. 380–385, August 2011. URL http://lis.csail.mit.edu/pubs/konidaris-aaai11a.pdf.

Lopez-Paz, David, Sra, Suvrit, Smola, Alex, Ghahramani, Zoubin, and Schölkopf, Bernhard. Randomized nonlinear component analysis. In *International Conference on Machine Learning (ICML)*, 2014.

Meier, Franziska, Hennig, Philipp, and Schaal, Stefan. Incremental local gaussian regression. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 972–980. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5594-incremental-local-gaussian-regression.pdf.

Nguyen-Tuong, Duy, Peters, Jan, Seeger, Matthias, and Schölkopf, Bernhard. Learning inverse dynamics: a comparison. In *European Symposium on Artificial Neural Networks*, 2008.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In Platt, J.C., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

Schaal, S. and Atkeson, C.G. Learning control in robotics. *Robotics Automation Magazine, IEEE*, 17(2):20–29, June 2010. ISSN 1070-9932. doi: 10.1109/MRA.2010.936957.

Shalev-Shwartz, Shai and Zhang, Tong. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14(1):567–599, 2013.

Smola, A. J. and Schölkopf, B. Sparse greedy matrix approximation for machine learning. In *Proceedings of the International Conference on Machine Learning*, pp. 911–918, San Francisco, 2000. Morgan Kaufmann Publishers.

Vijayakumar, Sethu and Schaal, Stefan. Locally weighted projection regression: An $O(n)$ algorithm for incremental real time learning in high dimensional space. In *Proc. Intl. Conf. Machine Learning*, pp. 1079–1086. Morgan Kaufmann, San Francisco, CA, 2000.

Williams, C. and Rasmussen, C. Gaussian processes for regression. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E. (eds.), *Advances in Neural Information Processing Systems 8*, volume 8, pp. 514–520, Cambridge, MA, 1996. MIT Press.

Williams, C. K. I. and Seeger, M. Using the Nystrom method to speed up kernel machines. In Dietterich, T. G., Becker, S., and Ghahramani, Z. (eds.), *Advances in Neural Information Processing Systems 14*, 2000.